An Analog Feedback Associative Memory

Amir Atiya, *Member, IEEE*, and Yaser S. Abu-Mostafa

Abstract— **Most of the neural network associative memory models deal with the storage of binary vectors. We consider the Hopfield continuous-time network, and develop a new method for the storage of analog vectors, i.e., vectors whose components are real-valued. An important requirement is that each memory vector has to be an asymptotically stable (i.e., attractive) equilibrium of the network. We point out some of the limitations of the continuous Hopfield model on the set of vectors that can be stored. These limitations can be relieved by choosing a network containing visible as well as hidden units. We have chosen an architecture consisting of several hidden layers and a visible layer, connected in a circular fashion. We prove that the two-layer case of such an architecture is guaranteed to store any number of given analog vectors provided their number does not exceed $1+$ the number of neurons in the hidden layer. We have developed a learning algorithm, which results in correctly adjusting the locations of the equilibria, as well as guaranteeing their asymptotic stability. Simulation results confirm the effectiveness of the new method.**

I. Introduction

ASSOCIATIVE memory has been an active research topic in the neural networks field. In one of the early papers, Hopfield [1] proposed a well-known associative memory model for binary vectors using a discrete feedback neural network. In this model every memory vector is an equilibrium of the network. When presented with an erroneous memory vector, the state of the network evolves to the equilibrium representing the correct memory vector. Afterwards many alternative techniques for storing binary vectors using also the discrete feedback network have been proposed, e.g., [2], [3], [4], and [5] (see also Michel and Farrell [6] for an overview). However, the problem of storing analog vectors, using Hopfield's *continuous* feedback network model of 1984 [7], has received little attention in literature. By analog vectors we mean vectors whose components are real-valued. This problem is important because in a variety of applications of associative memories like pattern recognition and vector quantization, the patterns are originally in analog form and therefore can save having the costly quantization step and therefore also save increasing the dimension of the vectors.

There are two main requirements for the associative memories. Every given memory should be an equilibrium of the network. Moreover, the equilibria corresponding to the memory vectors have to be asymptotically stable, i.e., they should be attractive. Previous work on the design of associative memories using Hopfield's continuous model include the work by Farrell and Michel [8]. Their model can store binary vectors in the form of asymptotically stable equilibria. Pineda [9] developed an interesting method for the storage of analog vectors. In this method the network is designed using a learning procedure so that each given memory vector becomes an equilibrium of the network. It does not guarantee though the asymptotic stability of the equilibria. In this paper we propose a method which solves the problem of storing analog vectors using the Hopfield continuous model. The network is designed so that each memory vector corresponds to an asymptotically stable equilibrium (see also a preliminary version of the method in Atiya and Abu-Mostafa [10]).

In the next section we will consider the continuous-time neural network model, study its storage limitations, and present the network architecture to be used. In Section III the learning algorithm will be developed. In Sections IV and V we consider extensions of the method to heteroassociative memory, and binary associative memory, respectively. In Section VI we derive the capacity of the memory. In Section VII implementation examples of the new method are demonstrated, and finally in Section VIII we summarize and give conclusions on the developed technique.

Manuscript received August 24, 1990; revised August 1, 1992. This work was supported by the Office of Naval Research under Grant N00014-89-J-1062.

A. Atiya was with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125. He is now with QANTXX Corp., Houston, TX 77029.

Y. Abu-Mostafa is with the Department of Electrical and Computer Science, California Institute of Technology, Pasadena, CA 91125.

IEEE Log Number 9204274.

II. The Network

Continuous-time neural networks can be modelled by the following differential equations (see Hopfield [7]):

$$\frac{d\boldsymbol{u}}{dt} = -\boldsymbol{u} + \boldsymbol{W}\boldsymbol{f}(\boldsymbol{u}) + \theta \equiv \boldsymbol{h}(\boldsymbol{u}). \qquad \boldsymbol{x} = \boldsymbol{f}(\boldsymbol{u}) \tag{1}$$

where $\boldsymbol{u} = (u_1, \cdots, u_N)^T$ is the vector of neuron potentials, $\boldsymbol{x} = (x_1, \cdots, x_N)^T$ is the vector of firing rates, $\boldsymbol{W}$ is the weight matrix, the (i,j)th element being w_{ij}, $\theta = (\theta_1, \cdots, \theta_N)^T$ is the threshold vector, $\boldsymbol{f}(\boldsymbol{u})$ means the vector $\left(f(u_1), \cdots, f(u_N)^T\right)$, f is a sigmoid-shaped function, for example

$$f(u) = \tanh(u) \tag{2}$$

and N is the number of neurons.

If the weight matrix is symmetric and f is monotonically increasing, then the state always converges to an equilibrium (Grossberg and Cohen [11] and Hopfield [7]). The vectors to be stored are set as equilibria of the network. Giving a noisy version of any of the stored vectors as the initial state of the network, the network state has to eventually reach the equilibrium state corresponding to the correct vector. An important requirement is that these equilibria are asymptotically stable,

otherwise no attraction to the equilibria will take place. By asymptotic stability of an equilibrium we mean that a trajectory starting in the vicinity of the equilibrium remains close and converges to it as $t \to \infty$ (see Guckenheimer and Holmes [12] for a formal definition, see also Michel *et al.* [13] and Salam *et al.* [14] for a study on asymptotic stability in continuous neural networks). Unfortunately, there is a limitation on the set of vectors which can be stored. We prove the following result.

Theorem 1: If $\boldsymbol{W}$ is symmetric and $f(u)$ is monotonically increasing, is strictly convex downwards for $u > 0$, and strictly convex upwards for $u < 0$, and $f(0) = 0$, then if $\boldsymbol{x}^*$ is a stored memory, then no other memory $\boldsymbol{y}^*$ can be stored with

$$|y_i^*| \geq |x_i^*|, \quad \operatorname{sign}(y_i^*) = \operatorname{sign}(x_i^*), \quad \text{all } i \text{ such that } x_i^* \neq 0 \tag{3}$$

or with

$$|y_i^*| \leq |x_i^*|, \qquad \operatorname{sign}(y_i^*) = \operatorname{sign}(x_i^*), \qquad \text{all } i, \tag{4}$$

where

$$\operatorname{sign}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0. \end{cases}$$

Proof: Assume contrary. First assume that there are two memories $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ satisfying (3). Both $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ are stored memories if and only if they are asymptotically stable equilibria of the system. We study the stability of the equilibrium $\boldsymbol{x}^*$ by linearizing (1) around the equilibrium (see Guckenheimer and Holmes [12]), to obtain

$$\frac{d\eta}{dt} = \boldsymbol{J}\eta$$

where $\eta = \boldsymbol{u} - \boldsymbol{u}^*$, $\boldsymbol{x}^* = \boldsymbol{f}(\boldsymbol{u}^*)$ and $\boldsymbol{J}$ is the Jacobian matrix, the (i,j)th element being $\partial h_i/\partial u_j$, evaluated at $\boldsymbol{u}^*$, where h_i is the ith element of $\boldsymbol{h}$ (of (1)). It can be evaluated as

$$\boldsymbol{J} = -\boldsymbol{I} + \boldsymbol{W}\boldsymbol{\Lambda}(\boldsymbol{u}^*)$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{\Lambda}(\boldsymbol{u}^*)$ is the diagonal matrix whose ith diagonal element is $f'(u_i^*)$. The equilibrium $\boldsymbol{u}^*$ in the nonlinear system (1) can be asymptotically stable only if $\boldsymbol{J}$ has eigenvalues of negative or zero real parts (see [12]). The eigenvalues of $\boldsymbol{J}$ are identical to those of $[-\boldsymbol{I} + \boldsymbol{\Lambda}^{1/2}(\boldsymbol{u}^*)\boldsymbol{W}\boldsymbol{\Lambda}^{1/2}(\boldsymbol{u}^*)]$ because if λ is an eigenvalue of $\boldsymbol{J}$, then

$$\det[-\boldsymbol{I} + \boldsymbol{W}\boldsymbol{\Lambda}(\boldsymbol{u}^*) - \lambda\boldsymbol{I}] = 0.$$

Upon multiplying both sides by $\det[\boldsymbol{\Lambda}^{1/2}(\boldsymbol{u}^*)]\det[\boldsymbol{\Lambda}^{-1/2}(\boldsymbol{u}^*)]$, we get

$$\det[-\boldsymbol{I} + \boldsymbol{\Lambda}^{1/2}(\boldsymbol{u}^*)\boldsymbol{W}\boldsymbol{\Lambda}^{1/2}(\boldsymbol{u}^*) - \lambda\boldsymbol{I}] = 0$$

using the fact that f is monotone increasing and hence $\boldsymbol{\Lambda}(\boldsymbol{u}^*)$ is nonsingular. The matrix $[-\boldsymbol{I} + \boldsymbol{\Lambda}^{1/2}(\boldsymbol{u}^*)\boldsymbol{W}\boldsymbol{\Lambda}^{1/2}(\boldsymbol{u}^*)]$ is negative semidefinite if and only if $[-\boldsymbol{\Lambda}^{-1}(\boldsymbol{u}^*) + \boldsymbol{W}]$ is negative semidefinite because for any vector z_1, $z_1^T[-\boldsymbol{\Lambda}^{-1}(\boldsymbol{u}^*) + \boldsymbol{W}]z_1$ equals $z_2^T[-\boldsymbol{I} + \boldsymbol{\Lambda}^{1/2}(\boldsymbol{u}^*)\boldsymbol{W}\boldsymbol{\Lambda}^{1/2}(\boldsymbol{u}^*)]z_2$, where $z_2 = \boldsymbol{\Lambda}^{-1/2}(\boldsymbol{u}^*)z_1$. By the mean value theorem, there exists a point $\boldsymbol{x}$ on the straight line joining $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ ($\boldsymbol{x} \neq \boldsymbol{x}^*, \boldsymbol{x} \neq \boldsymbol{y}^*$) such that

$$(y^* - x^*)^T h\big[f^{-1}(y^*)\big] - (y^* - x^*)^T h\big[f^{-1}(x^*)\big] = \left[\frac{\partial (y^* - x^*)^T h\big(f^{-1}(x)\big)}{\partial x}\right]^T (y^* - x^*)$$

(where $\boldsymbol{h}$ is as defined in (1)). But $\boldsymbol{h}\big[\boldsymbol{f}^{-1}(\boldsymbol{y}^*)\big] = \boldsymbol{h}\big[\boldsymbol{f}^{-1}(\boldsymbol{x}^*)\big] = 0$ since they are equilibria. We obtain

$$(\boldsymbol{y}^* - \boldsymbol{x}^*)^T[-\boldsymbol{\Lambda}^{-1}(\boldsymbol{u}) + \boldsymbol{W}](\boldsymbol{y}^* - \boldsymbol{x}^*) = 0 \tag{5}$$

where $\boldsymbol{u} = \boldsymbol{f}^{-1}(\boldsymbol{x})$. One observes that $\boldsymbol{x}$, being on the line joining $\boldsymbol{y}^*$ and $\boldsymbol{x}^*$, satisfies conditions

$$|x_i| \geq |x_i^*|, \quad \operatorname{sign}(x_i) = \operatorname{sign}(x_i^*), \quad \text{all } i \text{ such that } x_i^* \neq 0.$$

Because of the strict convexity assumption, $f'(u_i) < f'(u_i^*)$ when $|u_i| > |u_i^*|$. The left hand side of (5) can be written as

$$(\boldsymbol{y}^* - \boldsymbol{x}^*)^T[-\boldsymbol{\Lambda}^{-1}(\boldsymbol{u}) + \boldsymbol{W}](\boldsymbol{y}^* - \boldsymbol{x}^*) = -\sum_i \frac{(y_i^* - x_i^*)^2}{f'(u_i)} + \sum_i \sum_j w_{ij}(y_i^* - x_i^*)\big(y_j^* - x_j^*\big).$$

Whenever y_i^* is strictly larger than x_i^*, then $f'(u_i) < f'(u_i^*)$, and hence the right hand side of the previous equation is strictly less than

$$-\sum_i \frac{(y_i^* - x_i^*)^2}{f'(u_i^*)} + \sum_i \sum_j w_{ij}(y_i^* - x_i^*)\big(y_j^* - x_j^*\big)$$

which in turn is less than or equal to zero because the Jacobian at $\boldsymbol{u}^*$ has negative or zero eigenvalues. Hence (5) has no solution except when $\boldsymbol{x}^* = \boldsymbol{y}^*$, thus contradicting our assumption. The proof that there is no stable equilibrium $\boldsymbol{y}^*$ satisfying (4) follows by the argument that if this fact were not true, then by interchanging $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$, condition (3) would be satisfied, which we proved not to be possible. □

Fig. 1 illustrates the limitations given by Theorem 1 for a two-dimensional case. Fig. 1(a) considers the case when one of the memories $\boldsymbol{x}$ has no zero component. The "forbidden regions," or regions where no other memory can be stored, are shown to be the two rectangles below and to the inside, and above and to the outside the existing memory. Fig. 1(b) shows the case when one of the components of the existing memory is zero; then the forbidden region is now in general larger and extends over two quadrants. Fig. 1(c) illustrates an interesting corollary of the theorem, described as follows.

Corollary: If the origin is asymptotically stable then no other asymptotically stable equilibrium can exist.

The convexity requirements on f of Theorem 1 are satisfied by the most commonly used forms of sigmoid functions, like $f(u) = \tanh(u)$ and $f(u) = (2/\pi)\tan^{-1}(u)$. Even the absence of these requirements will only change the form but not necessarily the severity of the limitations.

We note that some of the attempts to characterize the set of asymptotically stable equilibria include the work by Li *et al.* [15]. However, their result, which states that two stable equilibria cannot coexist in a single quadrant, was shown to be wrong using a counter-example by Salam *et al.* [16].

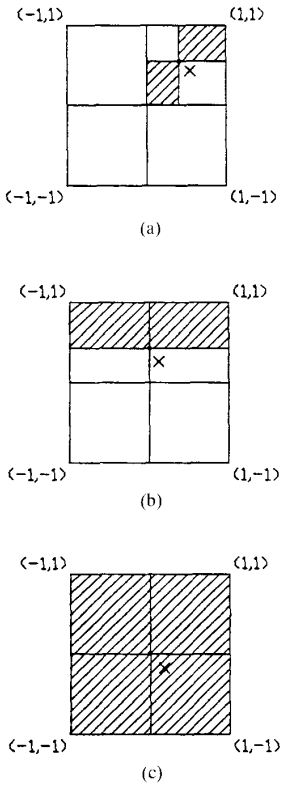

Fig. 1. An illustration of the limitations of the Hopfield continuous model on the set of memory vectors that can be stored. Consider a two-dimensional case. $\boldsymbol{x}$ represents one of the stored vectors. The hatched regions, or the "forbidden regions" represent the regions where no other memory can be stored. (a) Shows the case where $\boldsymbol{x}$ has no zero component, (b) shows the case when one of the components of $\boldsymbol{x}$ is zero, and (c) illustrates that storing the zero-vector prevents the ability to store any other vector.

The limitations given by Theorem 1 indicate some of the difficulties encountered in designing analog associative memories. There are sets of vectors which cannot be stored in the form of asymptotically stable equilibria. To solve this problem, we use an architecture consisting of both visible and hidden units. The outputs of the visible units correspond to the components of the stored vector. The purpose of the hidden units is to relieve the limitations on the visible components of the asymptotically stable equilibria. If the outputs of the hidden units are designed for example to have some components of a different sign for every different stored vector, then Theorem 1 does not provide any limitations on the freedom of choosing the sets of memory vectors to be stored (of course under the assumption of having enough hidden units). We will not restrict ourselves to a symmetric weight matrix. Our proposed model is as follows. We have a number of hidden layers and one visible layer, arranged in a circular fashion, with connections running in one direction from one layer to the next, as shown in Fig. 2. No connections exist within each of the layers. The advantage of such an architecture is that it lends itself to a simple design procedure, as we will see in the next section. It is worth noting that such an architecture has been considered in literature for various purposes. Atiya and

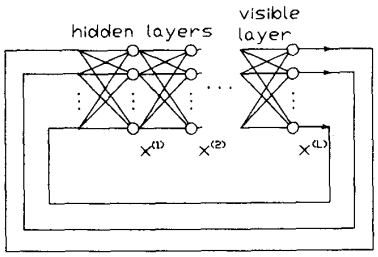

Fig. 2. The architecture proposed for the new associative memory model. We have one visible layer, and a number of hidden layers, connected in a circular fashion. Each memory vector corresponds to the visible layer component of an asymptotically stable equilibrium.

Baldi [17] analyzed oscillations and phase locking phenomena. Kosko [18] considered the special case of a two-layer network and proposed a model for heteroassociative memory for binary vectors called the Balanced Associative Memory (BAM), using discrete neurons.

Let $\boldsymbol{x}^{(l)}$ be the output vector of layer l. Then, our model is governed by the following set of differential equations,

$$\frac{d\boldsymbol{u}^{(l)}}{dt} = -\boldsymbol{u}^{(l)} + \boldsymbol{W}^{(l,l-1)}\boldsymbol{f}\left(\boldsymbol{u}^{(l-1)}\right) + \theta^{(l)},$$
$$\boldsymbol{x}^{(l)} = \boldsymbol{f}\left(\boldsymbol{u}^{(l)}\right) \tag{6}$$

where $\boldsymbol{W}^{(l,l-1)}$ is the $N_l \times N_{l-1}$ matrix of weights from layer $l-1$ to layer $l(\boldsymbol{W}^{(1,0)} \equiv \boldsymbol{W}^{(1,L)}, \boldsymbol{u}^{(0)} \equiv \boldsymbol{u}^{(L)}), \theta^{(l)}$ is the threshold vector for layer l, f is a sigmoid function in the range from -1 to 1, and L is the number of layers. The layers from 1 to $L-1$ represent the hidden layers, whereas layer L represents the visible layer.

We would like each memory vector to correspond to the visible layer component of an asymptotically stable equilibrium of the network. We have given a motivation for the choice of the architecture by saying that for this choice, Theorem 1 does not provide any limitations on the sets of memories which can be stored. However, it has to be shown that no limitations of other kind whatsoever will be encountered. The following theorem gives conditions for the stability of equilibria in the considered architecture.

Theorem 2: An equilibrium point

$$\boldsymbol{u}^* = \begin{pmatrix} \boldsymbol{u}^{*(1)} \\ \vdots \\ \boldsymbol{u}^{*(L)} \end{pmatrix}$$

satisfying

$$\sum_{j=1}^{N_l} |a_{ij}| f'\left(u_j^{*(l)}\right) < 1. \qquad i = 1, \ldots, N_l \tag{7}$$

for some layer l is asymptotically stable, where a_{ij} is the (i,j)th element of a matrix $\boldsymbol{A}$, given by

$$\boldsymbol{A} = \boldsymbol{W}^{(l,l-1)}\boldsymbol{\Lambda}\left(\boldsymbol{u}^{*(l-1)}\right)\ldots\boldsymbol{W}^{(2,1)}\boldsymbol{\Lambda}\left(\boldsymbol{u}^{*(1)}\right)\boldsymbol{W}^{(1,L)}$$
$$\cdot\,\boldsymbol{\Lambda}\left(\boldsymbol{u}^{*(L)}\right)\ldots\boldsymbol{\Lambda}\left(\boldsymbol{u}^{*(l-1)}\right)\boldsymbol{W}^{(l+1,l)}.$$

Proof: We linearize (6) around the equilibrium $\boldsymbol{u}^*$. We get

$$\frac{d\eta}{dt} = \boldsymbol{J}\eta$$

where $\eta = \boldsymbol{u} - \boldsymbol{u}^*$ and $\boldsymbol{J}$ is the Jacobian matrix, evaluated as in the equation shown at the bottom of the page. Let λ be an eigenvalue of $\boldsymbol{J}$, associated with eigenvector $\boldsymbol{v} = \left(\boldsymbol{v}_1^T, \ldots, \boldsymbol{v}_L^T\right)^T$, where $\boldsymbol{v}_l$ represents the component corresponding to layer l. Then,

$$\begin{aligned}
\boldsymbol{W}^{(1,L)}\boldsymbol{\Lambda}(\boldsymbol{u}^{*(L)})\boldsymbol{v}_L &= (\lambda+1)\boldsymbol{v}_1,\\
\boldsymbol{W}^{(2,1)}\boldsymbol{\Lambda}(\boldsymbol{u}^{*(1)})\boldsymbol{v}_1 &= (\lambda+1)\boldsymbol{v}_2.\\
&\vdots\\
\boldsymbol{W}^{(L,L-1)}\boldsymbol{\Lambda}(\boldsymbol{u}^{*(L-1)})\boldsymbol{v}_{L-1} &= (\lambda+1)\boldsymbol{v}_L,
\end{aligned}$$

from which we obtain

$$\boldsymbol{A}\boldsymbol{\Lambda}\left(\boldsymbol{u}^{*(l)}\right)\boldsymbol{v}_l = (\lambda+1)^L\boldsymbol{v}_l.$$

By Gershgorin's Theorem [19], any eigenvalue β of $\boldsymbol{A}\boldsymbol{\Lambda}\left(\boldsymbol{u}^{*(l)}\right)$ satisfies one of the following N_l inequalities:

$$|\beta| \le \sum_{j=1}^{N_l} |a_{ij}| f'\left(u_j^{*(l)}\right), \qquad i = 1, \ldots, N_l.$$

Since $\lambda = -1 + \beta^{1/L}$, then if Inequalities (7) are satisfied, λ will have a negative real part. Thus the equilibrium $\boldsymbol{u}^*$ of the system (6) is asymptotically stable. □

Thus if the neurons of one of the hidden layers are driven far enough into the saturation region, then the corresponding equilibrium will be stable because then, $f'(u_i^*)$ will be very small, causing Inequalities (7) to be satisfied. This provides a motivation for the learning algorithm, which is described next section.

III. Training the Network

Let $\boldsymbol{y}(m)$, $m = 1, \ldots, M$ be the vectors to be stored. Each $\boldsymbol{y}(m)$ corresponds to the visible layer component of an equilibrium. Let the equilibrium be denoted by $\boldsymbol{x}(m)$, where

$$\boldsymbol{x}(m) = \begin{pmatrix} \boldsymbol{x}^{(1)}(m) \\ \vdots \\ \boldsymbol{x}^{(L)}(m) \end{pmatrix}$$

where clearly $\boldsymbol{x}^{(L)}(m) = \boldsymbol{y}(m)$. We choose arbitrarily one of the hidden layers say layer K, and design the network such that the equilibrium values of that layer $x_i^{(K)}$, $i = 1, \ldots, N_K$ are very close to 1 or -1. This means the neurons in that layer are saturated, and we will therefore call this layer the saturating layer. Assuming that the outputs of the neurons in the saturating layer are close enough to 1 or -1, then Theorem 2 guarantees the asymptotic stability of the equilibrium. Enforcing the saturation of the units in the saturating layer is quite essential, since simulations show that when ignoring stability requirements, the equilibrium is probably unstable. Since we have an asymmetric network, there is nothing to rule out the existence of limit cycles. Also, we can have spurious equilibria, i.e., equilibria which correspond to no memory vector. However, if these unwanted limit cycles and spurious equilibria occur, then they would be far away from the memory vectors because each memory vector is guaranteed to have a basin of attraction around it.

The design of the memory is as follows. For each memory vector $\boldsymbol{y}(m)$ we choose an arbitrary target vector $\boldsymbol{z}(m)$ for the saturating layer, of the form $(\pm 1, \ldots, \pm 1)^T$. A simple way is to generate the $z_i(m)$'s randomly. We will call the $\boldsymbol{z}(m)$'s the *saturating layer vectors.*

Let us write the equations for the equilibrium of (6). Equating the right hand side to zero leads to:

$$\boldsymbol{x}^{(l)}(m) = \boldsymbol{f}\left(\boldsymbol{W}^{(l,l-1)}\boldsymbol{x}^{(l-1)}(m) + \theta^{(l)}\right), \qquad l = 1, \ldots, L. \tag{8}$$

The problem can be reduced to training an equivalent feedforward network (see Fig. (3)) to map input vector $\boldsymbol{y}(m)$ to output vector $\boldsymbol{y}(m)$ for $m = 1, \ldots, M$. This becomes clear if we compare the equations of the equilibrium for our feedback network (see (8)) with the equation for the output of the feedforward network:

$$\boldsymbol{y}^{(l)}(m) = \boldsymbol{f}\left(\boldsymbol{W}^{(l,l-1)}\boldsymbol{y}^{(l-1)}(m) + \theta^{(l)}\right), \qquad l = 1, \ldots, L$$

when the input $\boldsymbol{y}^{(0)}(m)$ is $\boldsymbol{y}(m)$ ($\boldsymbol{y}^{(l)}(m)$ is the output of layer l for the feedforward network).

We define the two error functions E_1 and E_2,

$$\begin{aligned}
E_1 &= \sum_{m=1}^{M} E_1(m), \qquad E_1(m) = \left\|\boldsymbol{y}^{(K)}(m) - \boldsymbol{z}(m)\right\|^2\\
E_2 &= \sum_{m=1}^{M} E_2(m), \qquad E_2(m) = \left\|\boldsymbol{y}^{(L)}(m) - \boldsymbol{y}(m)\right\|^2.
\end{aligned}$$

Training is performed by applying a backpropagation type algorithm [20] twice. First we train layers 1 to K to map the input vector $\boldsymbol{y}(m)$ to the saturating layer vector $\boldsymbol{z}(m)$, by minimizing E_1 for the feedforward network. Then, we train layers $K+1$ to L to map the output of the saturating layer K to the output vector $\boldsymbol{y}(m)$, by minimizing E_2.

$$\boldsymbol{J} = -\boldsymbol{I} + \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{W}^{(1,L)}\boldsymbol{\Lambda}\left(\boldsymbol{u}^{*(L)}\right) \\ \boldsymbol{W}^{(2,1)}\boldsymbol{\Lambda}\left(\boldsymbol{u}^{*(1)}\right) & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{W}^{(L,L-1)}\boldsymbol{\Lambda}\left(\boldsymbol{u}^{*(L-1)}\right) & \boldsymbol{0} \end{pmatrix}.$$

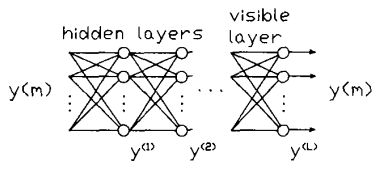

Fig. 3. The equivalent feedforward network.

We note that we use a slightly modified version of the backpropagation algorithm for training the layers 1 to K. A standard backpropagation algorithm will have a very slow convergence because the targets are 1 or -1. Therefore, in our method we used a stopping criterion as follows. If the output signs match the target signs and the outputs are at least some constant β in magnitude (a typical value $\beta = 0.8$), then we stop iterating the weights. We then multiply the weights and the thresholds of the saturating layer by a big positive constant in order to drive the outputs of the saturating layer as close as possible to 1 or -1. Let ϵ_1 and ϵ_2 be two small positive constants (typical values: $\epsilon_1 = 0.01$ and $\epsilon_2 = 10^{-5}$), and let ρ denote the step size. Our algorithm is as follows.

1) Generate the initial setting of the weight matrices $\boldsymbol{W}^{(l,l-1)}(0)$ and $\theta^{(l)}(0)$ randomly, $l = 1, \ldots, L$.
2) Apply the steepest descent update (the delta rule of [17]) for E_1 for the first K layers,

$$\boldsymbol{W}^{(l,l-1)}(n) = \boldsymbol{W}^{(l,l-1)}(n-1) - \rho \frac{\partial E_1}{\partial \boldsymbol{W}^{(l,l-1)}}, \quad l = 1, \ldots, K$$

$$\theta^{(l)}(n) = \theta^{(l)}(n-1) - \rho \frac{\partial E_1}{\partial \theta^{(l)}}, \quad l = 1, \ldots, K$$

until $\operatorname{sgn}[y_i^{(K)}(m)] = \operatorname{sgn}[z_i(m)]$, and $|y_i^{(K)}(m)| \geq \beta$ for all i and m.
3) Set

$$\boldsymbol{W}^{(K,K-1)} = c\boldsymbol{W}^{(K,K-1)}(n)$$
$$\theta^{K} = c\theta^{(K)}(n)$$

where c is a positive constant such that $|y_i^{(K)}(m)| \geq 1 - \epsilon_1$ for all i and m.
4) Apply the steepest descent update for E_2 for layers $K+1$ to L,

$$\boldsymbol{W}^{(l,l-1)}(n) = \boldsymbol{W}^{(l,l-1)}(n-1) - \rho \frac{\partial E_2}{\partial \boldsymbol{W}^{(l,l-1)}}, \quad l = K+1, \ldots, L$$

$$\theta^{(l)}(n) = \theta^{(l)}(n-1) - \rho \frac{\partial E_2}{\partial \theta^{(l)}}, \quad l = K+1, \ldots, L$$

until $E_2 < \epsilon_2$.

Remarks:

a) It is important to note that for training layers $K+1$ to L, the input vector to layer $K+1$ is of course taken to be the output vector of layer K, not the target vector $\boldsymbol{z}(m)$ of the saturating layer, because they are slightly different.
b) After learning has been completed, the stability of the equilibria has to be checked by simulating the set of differential equations, or else by checking Inequalities (7). If some of the equilibria are unstable, then this means that we have not chosen a small enough ϵ_1. In our simulations we have never encountered an unstable equilibrium when $\epsilon_1 = 0.02$.
c) Another frequently used variant of the backpropagation algorithm, for which we have obtained generally faster convergence here, is to update the weights after presenting each training input instead of after cycling through all the training inputs.
d) Let $\boldsymbol{W}$ and θ represent respectively the weight matrix and threshold vector at which (8) is satisfied. Practically, the learning algorithm might result in an output of the feedforward network slightly different from $\boldsymbol{y}(m)$, when inputting $\boldsymbol{y}(m)$, and a parameter set $\boldsymbol{W}'$ and θ' slightly differ from $\boldsymbol{W}$ and θ, respectively. We assert that this produces only a slight displacement of the equilibrium of the feedback network. This is because the equilibrium is stable and hence the corresponding Jacobian is nonsingular (for example there will be no bifurcations at which the equilibrium might disappear).
e) A very important remark: Once learning is completed, retrieving a memory vector $\boldsymbol{y}(m)$ using a noisy version $\boldsymbol{y}'(m)$ is performed as follows. The initial condition for the visible layer is set as $\boldsymbol{y}'(m)$. The initial conditions for the remaining layers are set as

$$\boldsymbol{x}^{(l)} = \boldsymbol{f}\left(\boldsymbol{W}^{(l,l-1)}\boldsymbol{x}^{(l-1)} + \theta^{(l)}\right), \quad l = 1, \ldots, L-1$$

where $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(L)} \equiv \boldsymbol{y}'(m)$. A practical way to load the initial conditions is by clamping $\boldsymbol{y}'(m)$ shortly onto the visible layer and allowing the network to evolve according to its equations of motion (6), while the connections from the last hidden layer to the visible layer are temporarily switched off.
f) The presented training method applies also to the discrete-time continuous-output update formulation, i.e., the system given by

$$\boldsymbol{x}^{(l)}(t+1) = \boldsymbol{f}\left(\boldsymbol{W}^{(l,l-1)}\boldsymbol{x}^{(l-1)}(t) + \theta^{(l)}\right), \quad l = 1, \ldots, L.$$

The equilibria for such a system are equivalent to those of the continuous-time system (6). Also, we can show using a theorem analogous to Theorem 2 that if one of the hidden layers is heavily saturated, then the corresponding equilibrium is asymptotically stable. This system has yet to be tested by numerical simulations, to verify its global stability properties.

IV. Heteroassociative Memory

Any auto-associative memory can be easily extended into a heteroassociative memory by applying an appropriate linear transformation at the output. We present a more direct way. Let $(\boldsymbol{c}(1), \boldsymbol{y}(1)), \ldots, (\boldsymbol{c}(M), \boldsymbol{y}(M))$ be the pairs of stored associations, and $\boldsymbol{z}(1), \ldots, \boldsymbol{z}(M)$ be the corresponding saturating layer vectors. We have at least 3 layers in our loop architecture: one saturating layer K_1, and two layers K_2 and L, which upon convergence are to carry $\boldsymbol{c}(m)$ and $\boldsymbol{y}(m)$, respectively. In a way, this is a multilayer analog extension of Kosko's two-layer heteroassociative BAM [18]. Consider also an equivalent feedforward network as in the previous section. Let

$$E_1 = \sum_{m=1}^{M} E_1(m), \qquad E_1(m) = \left\| \boldsymbol{y}^{(K_1)}(m) - \boldsymbol{z}(m) \right\|^2$$

$$E_2 = \sum_{m=1}^{M} E_2(m), \qquad E_2(m) = \left\| \boldsymbol{y}^{(K_2)}(m) - \boldsymbol{c}(m) \right\|^2$$

$$E_3 = \sum_{m=1}^{M} E_3(m), \qquad E_3(m) = \left\| \boldsymbol{y}^{(L)}(m) - \boldsymbol{y}(m) \right\|^2.$$

Learning is performed according to the following straightforward extension of the learning algorithm of the previous section.

1) Generate the initial setting of the weight matrices $\boldsymbol{W}^{(l,l-1)}(0)$ and $\theta^{(l)}(0)$ randomly, $l = 1, \ldots, L$.
2) Apply the steepest descent update for E_1 for the first K_1 layers,

$$\boldsymbol{W}^{(l,l-1)}(n) = \boldsymbol{W}^{(l,l-1)}(n-1) - \rho \frac{\partial E_1}{\partial \boldsymbol{W}^{(l,l-1)}}, \quad l = 1, \ldots, K_1$$

$$\theta^{(l)}(n) = \theta^{(l)}(n-1) - \rho \frac{\partial E_1}{\partial \theta^{(l)}}, \quad l = 1, \ldots, K_1$$

until $\operatorname{sign}[y_i^{(K_1)}(m)] = \operatorname{sign}[z_i(m)]$, and $|y_i^{(K_1)}(m)| \geq \beta$ for all i and m.
3) Set

$$\boldsymbol{W}^{(K_1,K_1-1)} = c\boldsymbol{W}^{(K_1,K_1-1)}(n)$$
$$\theta^{(K_1)} = c\theta^{(K_1)}(n)$$

where c is a positive constant such that $|y_i^{(K_1)}(m)| \geq 1 - \epsilon_1$ for all i and m.
4) Apply the steepest descent update for E_2 for layers $K_1 + 1$ to K_2,

$$\boldsymbol{W}^{(l,l-1)}(n) = \boldsymbol{W}^{(l,l-1)}(n-1) - \rho \frac{\partial E_2}{\partial \boldsymbol{W}^{(l,l-1)}}, \quad l = K_1 + 1, \ldots, K_2$$

$$\theta^{(l)}(n) = \theta^{(l)}(n-1) - \rho \frac{\partial E_2}{\partial \theta^{(l)}}, \quad l = K_1 + 1, \ldots, K_2$$

until $E_2 < \epsilon_2$.
5) Apply the steepest descent update for E_3 for layers $K_2 + 1$ to L,

$$\boldsymbol{W}^{(l,l-1)}(n) = \boldsymbol{W}^{(l,l-1)}(n-1) - \rho \frac{\partial E_3}{\partial \boldsymbol{W}^{(l,l-1)}}, \quad l = K_2 + 1, \ldots, L$$

$$\theta^{(l)}(n) = \theta^{(l)}(n-1) - \rho \frac{\partial E_3}{\partial \theta^{(l)}}, \quad l = K_2 + 1, \ldots, L$$

until $E_3 < \epsilon_2$.

V. A Binary Associative Memory

In case of a binary associative memory, a method simpler than the one presented in Section III is even possible. Let the memory vectors $\boldsymbol{y}(1), \ldots, \boldsymbol{y}(M)$ be of the form $(\pm 1, \ldots, \pm 1)^T$. At an equilibrium corresponding to some memory vector $\boldsymbol{y}(m)$, the visible layer is heavily saturated, and therefore by Theorem 2 the equilibrium is stable, even if we do not have a saturating hidden layer. A saturating hidden layer is therefore not needed and the problem reduces to applying a backpropagation algorithm only once on the feedforward network of Fig. 3. The set of input vectors are $\boldsymbol{y}(1), \ldots, \boldsymbol{y}(M)$, and the corresponding target output vectors at the last layer L are again $\boldsymbol{y}(1), \ldots, \boldsymbol{y}(M)$. Let

$$E_B = \sum_{m=1}^{M} \left\| \boldsymbol{y}^{(L)}(m) - \boldsymbol{y}(m) \right\|^2.$$

The algorithm is as follows.

1) Generate the initial setting of the weight matrices $\boldsymbol{W}^{(l,l-1)}(0)$ and $\theta^{(l)}(0)$ randomly, $l = 1, \ldots, L$.
2) Apply the steepest descent update for E_B for all L layers,

$$\boldsymbol{W}^{(l,l-1)}(n) = \boldsymbol{W}^{(l,l-1)}(n-1) - \rho \frac{\partial E_B}{\partial \boldsymbol{W}^{(l,l-1)}}, \quad l = 1, \ldots, L$$

$$\theta^{(l)}(n) = \theta^{(l)}(n-1) - \rho \frac{\partial E_B}{\partial \theta^{(l)}}, \qquad l = 1, \ldots, L$$

until $\operatorname{sign}[y_i^{(L)}(m)] = \operatorname{sign}[z_i(m)]$, and $|y_i^{(L)}(m)| \geq \beta$ for all i and m.
3) Set

$$\boldsymbol{W}^{(L,L-1)} = c\boldsymbol{W}^{(L,L-1)}(n)$$
$$\theta^{(L)} = c\theta^{(L)}(n)$$

where c is a positive constant such that $|y_i^{(L)}(m)| \geq 1 - \epsilon_1$ for all i and m.

VI. The Capacity of the Network

A. On the Capacity of Continuous-Time Networks

For the Hopfield network, described by (1), it was shown by Guez *et al.* [21] that there are at most 3^N equilibria. One can show that one can obtain 2^N asymptotically stable equilibria by considering a diagonal weight matrix with sufficiently large entries. For the architecture considered, it is possible

to show that the number of asymptotically stable equilibria one can achieve is at least 2 to the power the number of neurons in the smallest layer. However, the issue here is not so much how may equilibria one can have, as much as how programmable are their locations. Consider a general fully connected continuous Hopfield network (see (1)) with I visible neurons and J hidden neurons. The network should work as an associative memory, in the sense that each memory vector $\boldsymbol{y}^{(m)}$ corresponds to an equilibrium. Let M be the number of memory vectors, and N be the total number of neurons (i.e., $N = I + J$). Then we have MI equations for the equilibria, while there are $(I + J)(I + J + 1)$ degrees of freedom, corresponding to the weights and the thresholds. Thus the capacity is at most

$$\frac{(I + J)(I + J + 1)}{I}. \tag{9}$$

The set of vectors which can be stored if M is greater than (9) represents the range of the mapping which maps $(\boldsymbol{W}, \theta)$ to $(\boldsymbol{y}(1), \ldots, \boldsymbol{y}(M))$ if $\boldsymbol{y}(1), \ldots, \boldsymbol{y}(M)$ are stable equilibria of the network, and therefore it is a number of surfaces of a lower dimension embedded in the space of all possible sets of M memory vectors. Therefore, if M is greater than (9), then the fraction of all possible sets of vectors which can be stored is virtually a set of measure zero.

Consider a fully connected network with no hidden neurons. Apart from the constraint shown in Theorem 1 on the set of vectors which can be stored, it follows from the previous argument that only a zero-fraction of the set of all possible M vectors can be stored, if $M > N + 1$. Thus there is an abrupt change when M exceeds $N+1$. In case of a symmetric matrix the abrupt change occurs at $(N + 3)/2$.

B. The Capacity of the Proposed Model

An expression for the capacity is difficult to obtain for the general multilayer network. The two-layer case is of special interest because of its simplicity and the adequacy of its capabilities. Let N_1 and N_2 be the numbers of neurons in the hidden and the visible layers, respectively. Using an argument similar to the one presented in the previous subsection, we can conclude that the capacity cannot exceed $(2N_1N_2 + N_1 + N_2)/N_2$, which is approximately $2N_1$ for large N_1 and N_2. Of more interest is to obtain a lower bound on the capacity, since one is interested more in what the network can do rather than what it cannot do. The following theorem gives a lower bound. First define the *augmented memory vectors* and the *augmented saturating layer vectors* as respectively $[\boldsymbol{y}(1)^T \mid 1]^T, \ldots, [\boldsymbol{y}(M)^T \mid 1]^T$ and $[\boldsymbol{z}(1)^T \mid 1]^T, \ldots, [\boldsymbol{z}(M)^T \mid 1]^T$.

Theorem 3: A two-layer network is guaranteed to store any set of $N_1 + 1$ vectors.

Proof: Sufficient conditions for the storage of the memory vectors are:

$$\left[\boldsymbol{W}^{(2,1)} \mid \theta^{(2)}\right] \boldsymbol{Z} = \boldsymbol{V} \tag{10}$$

$$\left[\boldsymbol{W}^{(1,2)} \mid \theta^{(1)}\right] \boldsymbol{Y} = \boldsymbol{F} \tag{11}$$

and

$$\sum_{j=1}^{N_1} |a_{ij}| f'(u_j(m)) < 1. \qquad i = 1, \ldots, N_1 \tag{12}$$

where $\boldsymbol{Z}$ is the matrix whose columns are the augmented saturating layer vectors, $\boldsymbol{Y}$ is the matrix whose columns are the augmented memory vectors, $\boldsymbol{V}$ and $\boldsymbol{F}$ are the matrices given by

$$\boldsymbol{V} = \begin{pmatrix} f^{-1}[y_1(1)] & \cdots & f^{-1}[y_1(M)] \\ \vdots & \vdots & \vdots \\ f^{-1}[y_{N_2}(1)] & \cdots & f^{-1}[y_{N_2}(M)] \end{pmatrix}.$$

$$\boldsymbol{F} = \begin{pmatrix} f^{-1}[z_1(1)] & \cdots & f^{-1}[z_1(M)] \\ \vdots & \vdots & \vdots \\ f^{-1}[z_{N_1}(1)] & \cdots & f^{-1}[z_{N_1}(M)] \end{pmatrix}.$$

$u_j(m) = f^{-1}(z_j(m))$, and a_{ij} is the (i, j)th element of the matrix $\boldsymbol{A}$, defined as $\boldsymbol{A} = \boldsymbol{W}^{(1,2)} \boldsymbol{\Lambda}[f^{-1}(\boldsymbol{y}(m))] \boldsymbol{W}^{(2,1)}$. Equations (10) and (11) guarantee that the memory vectors are equilibria, whereas Inequalities (12) ensure their stability, according to Theorem 2.

Let $\boldsymbol{Z}$ be given by the following $(N_1 + 1) \times M$ matrix:

$$\boldsymbol{Z} = \begin{pmatrix} 1 & -1 & -1 & \cdots & -1 \\ 1 & 1 & -1 & \cdots & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & -1 \\ 1 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix}.$$

Equation (10) represents N_2 sets of linear equations in the weights and the thresholds. A solution for $[\boldsymbol{W}^{(2,1)} \mid \theta^{(2)}]$ exists if $M \leq N_1 + 1$, since the columns of $\boldsymbol{Z}$ are linearly independent. Consider (11). It is enough to prove that there is a matrix $\boldsymbol{W}$ and a threshold vector θ satisfying

$$\operatorname{sign}[\boldsymbol{W}\boldsymbol{y}(m) + \theta]_i = \operatorname{sign}[z_i(m)], \qquad \text{all } i, m \tag{13}$$

because one can then take $\boldsymbol{W}^{(1,2)} = \alpha \boldsymbol{W}$, and $\theta^{(1)} = \alpha\theta$, where α is big enough to drive the outputs of the hidden units far into the saturation region, i.e., very close to -1 or 1. It is possible to find a $\boldsymbol{W}$ and θ to satisfy (13), as follows. One can take $\boldsymbol{W} = (\boldsymbol{w} | \ldots | \boldsymbol{w})^T$ where $\boldsymbol{w}$ is a column vector satisfying

$$\boldsymbol{w}^T \boldsymbol{y}(i) \neq \boldsymbol{w}^T \boldsymbol{y}(j). \qquad \text{for } i \neq j. \tag{14}$$

Such a $\boldsymbol{w}$ exists for any set of distinct vectors $\boldsymbol{y}(1), \ldots, \boldsymbol{y}(M)$ because otherwise $\boldsymbol{w}$ can be perturbed to restore (14). Rename the indices of the memories such that

$$\boldsymbol{w}^T \boldsymbol{y}(1) > \boldsymbol{w}^T \boldsymbol{y}(2) > \ldots > \boldsymbol{w}^T \boldsymbol{y}(M).$$

We choose the θ_i's such that

$$\boldsymbol{w}^T \boldsymbol{y}(1) > -\theta_1 > \boldsymbol{w}^T \boldsymbol{y}(2) > -\theta_2 > \ldots > \boldsymbol{w}^T \boldsymbol{y}(M) > -\theta_M > \ldots > \theta_{N_1}.$$

One can see that this choice satisfies (13). Now we show that taking α large enough will satisfy Inequalities (12). We take the limit as $\alpha \to \infty$. We get

$$\lim_{\alpha\to\infty} \sum_{j=1}^{N_1} |a_{ij}|\alpha f'(\alpha u_j(m)) = \lim_{\alpha\to\infty} \sum_{j=1}^{N_1} |a_{ij}|\alpha \frac{4e^{-2\alpha u_j(m)}}{\left(1+e^{-2\alpha u_j(m)}\right)^2} = 0 < 1$$

where we used the sigmoid function defined in (2). This completes the proof that $N_1 + 1$ memory vectors correspond to asymptotically stable equilibrium points. □

Thus, the storage capacity equals 1 + number of hidden neurons. For the network to achieve this capacity, a particular choice of the saturating layer vectors has to be taken. In our algorithm the saturating layer vectors are specified by the user, as generating them randomly for example realizes the distributiveness of the memory as well as results in good retrieval accuracy. The question is whether for any choice of saturating layer vectors the capacity will remain the same as $N_1 + 1$. The answer turns out to be negative, and the capacity can be lower in this case, as will be shown in the next theorem.

Theorem 4: If the augmented memory vectors are linearly independent, and the augmented saturating layer vectors are linearly independent, then a two layer network is guaranteed to store $\min(N_1, N_2) + 1$ vectors.

Proof: The memory vectors correspond to the stable equilibria if conditions (10)–(12) are satisfied. Equation (10) has a solution for $\boldsymbol{W}^{(2,1)}$ since $M \leq N_1 + 1$ and the columns of $\boldsymbol{Z}$ are linearly independent by hypothesis. Also, (11) has a solution for $\boldsymbol{W}^{(1,2)}$ because $M \leq N_2 + 1$ and the columns of $\boldsymbol{Y}$ are linearly independent, again by the hypothesis of the theorem. Inequalities (12) are satisfied by choosing the magnitudes of the elements of $\boldsymbol{F}$ large enough and using an argument similar to that in the proof of Theorem 3. □

If $M > \min(N_1, N_2) + 1$, then (10) and (11) will constitute more equations than unknowns. Hence, with a random choice of the saturating layer vectors, there is a small chance for being able to store a given set of vectors. We must caution however that the previous theorem gives only the number of vectors which can be stored when the saturating layer vectors are given. It does not guarantee their storage when using the learning algorithm presented in Section III. This is because it is possible to get stuck in a local minimum of the error function. Although there is a simple way to satisfy the conditions (10)–(12) sufficient for the storage of the memory vectors by simply solving the linear equations (10) and (11), we favor using the learning method for the following reason. If we are not at full capacity, then (10) and (11) do not have a unique solution. Specifying a solution arbitrarily by for example randomly generating the values of the extra degrees of freedom has led to results quite inferior to those of the learning method.

VII. Implementation

The theoretical results presented in the last section indicate some of the capabilities of the new associative memory. However, some of the issues remain yet unresolved by the theoretical analysis, like the extent of the existence of limit cycles and spurious equilibria, and the shapes of the basins of attraction. We have therefore resorted to numerical simulations. In the numerous experiments we have performed we have never encountered limit cycles. Spurious equilibria exist, but their number is high only when we are close to full capacity. Concerning the basins of attraction, they are required to have balanced shapes (for example no memory vector has a too small or a too large basin of attraction). We have tested this by performing the following experiment. We have a two-layer network with two hidden neurons and two visible neurons. The two vectors $(-0.5, -0.5)^T$ and $(0.75, 0.25)^T$ are stored. Fig. 4 shows the resulting basins of attraction.

To test the error correction capability of the model, we have performed the following experiment. A two-layer network is considered with 10 neurons in the hidden layer and 10 neurons in the visible layer. Five randomly generated memory vectors are considered, with components ranging from -0.8 to 0.8 (if we have components close to 1 or -1, then convergence in learning can get very slow because we are close to the saturation region). Thus, we are operating at about half capacity. After learning, the model is tested by giving memory vectors corrupted by noise (100 vectors for a given variance). Fig. 5 shows the error fraction versus the signal to noise ratio. The model is also tested for pattern completion ability. Components of the patterns are chosen randomly and set to zero. Fig. 6 shows the error fraction versus the number of missing components. The capacity according to Theorem 4 for the two-layer network is 11. We have done several experiments with 11 memory vectors. In only a few instances did learning succeed. The reason is partly the fact that the augmented saturation layer vectors turn out to be linearly dependent, and partly the failure of the learning algorithm to converge to the global minimum. We have done also several experiments with 10 memory vectors. In all our trials learning succeeded. However, the results with 10 memory vectors are much worse than those of the case of 5 memory vectors. The main reason is that the number of spurious equilibria in the 10 vector case is larger. We have observed in general that when operating close to the full capacity, the number of spurious equilibria increases. A reasonable strategy is therefore to operate at some fraction of the full capacity. We performed another experiment using a four-layer network. We have 10 neurons per layer, and 5 memory vectors. The results are shown in Fig. 7 for the case of having noisy memory vectors, and in Fig. 8 for the case of having memory vectors with missing components. We observe that the results are close to those of the two layer network. For high signal to noise ratios, the two-layer network achieves higher recall accuracy, whereas for low signal to noise ratios the four layer network performs better. We were able to store up to 13 memory vectors using the four layer network. As in the case of two layer network, close to the capacity the performance is not very good.

Finally, the storage capacity of the network for binary vectors is tested. We implemented the method described in Section V. A two-layer network is considered, having 10 neurons in each layer. Of course in this case there is no saturating

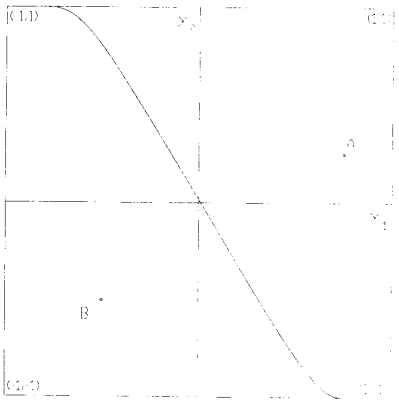

Fig. 4. A demonstration of the basins of attraction for a network with two hidden neurons and two visible neurons, after training the network to store the vectors $A = (0.25, 0.75)^T$ and $B = (-0.5, -0.5)^T$.

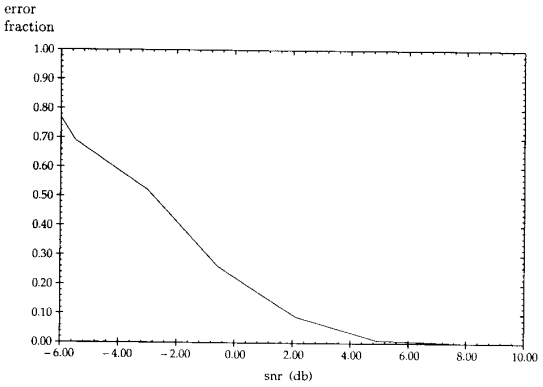

Fig. 5. The error fraction versus the signal to noise ratio for a two layer network with 10 neurons per layer. The signal to noise ratio here means 10 times the logarithm of ratio of the average of the square of the vector components and the noise variance.

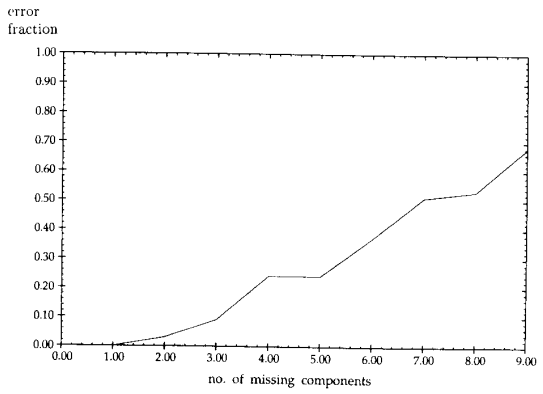

Fig. 6. The error fraction versus the number of missing components for a two-layer network with 10 neurons per layer.

hidden layer. Again, 5 memory vectors are considered. The pattern completion ability of the network is tested, and Fig. 9 shows the result. We observe that the performance is better than the case of storing real valued vectors, due to the fact that

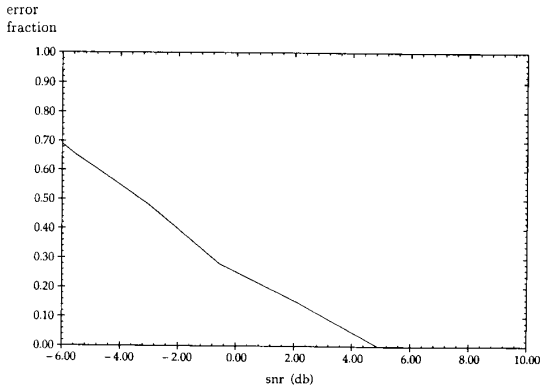

Fig. 7. The error fraction versus the signal to noise ratio for a four layer network with 10 neurons per layer.

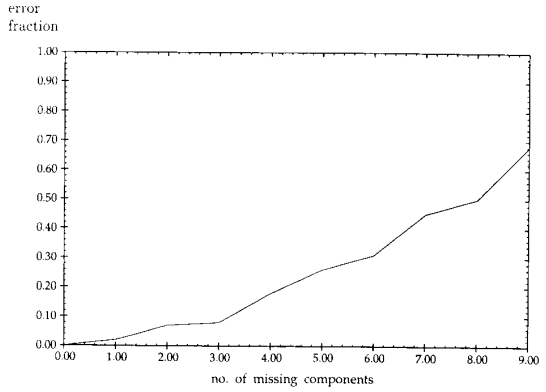

Fig. 8. The error fraction versus the number of missing components for a four-layer network with 10 neurons per layer.

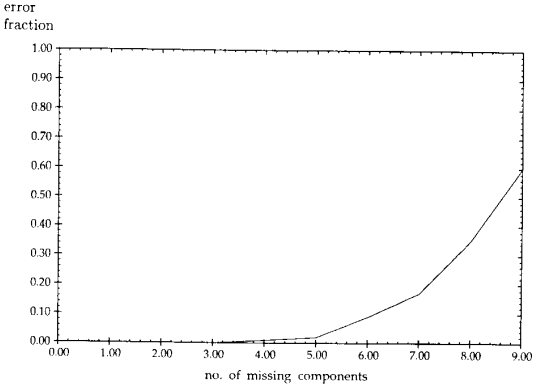

Fig. 9. The error fraction versus the number of missing bits for a two-layer network with 10 neurons per layer, storing binary vectors.

the case of binary vectors is to a greater extent below its capacity, which is here much higher than the case of analog vectors.

VIII. Conclusions

We have developed a new associative memory model for real valued vectors, using the continuous feedback neural

network model. We have considered an architecture consisting of a visible layer and a number of hidden layers connected in a circular fashion. Training is reduced to the problem of training a feedforward network and therefore the standard backpropagation routines can be used.

The proposed associative memory is of a distributed nature and therefore a localized fault in one of the neurons will not necessarily "erase" one of the memories, unlike some of the associative memories for instance those of the winner-take-all type (see Lippmann's Hamming Net [22], and Majani *et al.* [23]). The strength of the new method is that it can cope with constraints on the architecture, like fan-in and fan-out limitations, or locality of interconnections. In contrast, most of the other associative memory models do not possess such a flexibility, and for example they require full connectivity. Another advantage of the new memory is that it can store any given set of vectors. For most of the distributed associative memories (i.e., not counting winner-take-all type), like those using the Hopfield binary model, there are sets of vectors which cannot be stored (see [24]).

REFERENCES

[1] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," in *Proc. Nat. Acad. Sci. USA*, vol. 79, 1982, pp. 2554–2558.

[2] S. Venkatesh and D. Psaltis, "Linear and logarithmic capacities in associative neural networks," *IEEE Informat. Theory*, vol. 35, pp. 558–568, 1989.

[3] L. Personnaz, I. Guyon, and G. Dreyfus, "Information storage and retrieval in spin-glass like neural networks," *J. Phys. Lett.*, vol. 46, pp. L359–L365.

[4] T. Chiueh and R. Goodman, "A neural network classifier based on coding theory," in *Neural Information Processing Systems*, D. Anderson, Ed. New York: American Institute of Physics, 1988, pp. 174–183.

[5] J.-W. Wong, "Recognition of general patterns using neural networks," *Biological Cybernetics*, vol. 58, no. 6, pp. 361–372, 1988.

[6] A. Michel and J. Farrell, "Associative memories via artificial neural networks," *IEEE Contr. Syst. Mag.*, pp. 6–17, Apr. 1990.

[7] J. Hopfield, "Neurons with graded response have collective computational properties like those of two state neurons," in *Proc. Nat. Acad. Sci. USA*, vol. 81, 1984, p. 3088–3092.

[8] J. Farrell and A. Michel, "A synthesis procedure for Hopfield's continuous-time associative memory," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 877–884, 1990.

[9] F. Pineda, "Dynamics and architecture in neural computation," *J. Complexity*, Sept. 1988, to be published.

[10] A. Atiya and Y. Abu-Mostafa, "A method for the associative storage of analog vectors," in *Advances in Neural Information Processing Systems 2*, D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 590–595.

[11] S. Grossberg and M. Cohen, "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, pp. 815–826, 1983.

[12] J. Guckenheimer and R. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields.* New York: Springer-Verlag, 1983.

[13] A. Michel, J. Farrell and W. Porod, "Qualitative analysis of neural networks," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 229–243, 1989.

[14] F. Salam, Y. Wang, and M-R. Choi, "On the analysis of dynamic feedback neural nets," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 196–201, 1991.

[15] J. Li, A. Michel, and W. Porod, "Qualitative analysis and synthesis of a class of neural networks," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 976–986, 1988.

[16] F. Salam, and Y. Wang, "Some properties of dynamic feedback neural nets," in *Proc. 27th Conf. Decision and Control*, Austin, TX, Dec. 1988, pp. 337–342.

[17] A. Atiya and P. Baldi, "Oscillations and synchronization in neural networks: an exploration of the labeling hypothesis," *Int. J. Neural Syst.*, vol. 1, no. 2, pp. 103–124, 1989.

[18] B. Kosko, "Bidirectional associative memories," *IEEE Trans. Syst., Man, Cybern.*, vol. 18, pp. 49–60, 1988.

[19] J. Franklin, *Matrix Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1968.

[20] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. Rumelhart and J. McClelland, Eds. Cambridge, MA: MIT Press, 1986.

[21] A. Guez, V. Protopopsecu, and J. Barhen, "On the stability, storage capacity, and design of nonlinear continuous neural networks," *IEEE Trans. Syst., Man, Cybern.*, vol. 18, pp. 80–87, 1988.

[22] R. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4–22, 1987.

[23] E. Majani, R. Erlanson, and Y. Abu-Mostafa, "On the K-winners-take-all network," in *Advances in Neural Information Processing Systems I*, D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1988, pp. 634–642.

[24] J. Bruck and J. Sanz, "A study on neural networks," *Int. J. Intelligent Syst.*, vol. 3, pp. 59–75, 1988.

Amir Atiya (S'86–M'90) was born in Cairo, Egypt, on March 20, 1960. He received the B.S. degree in 1982 from Cairo University, Cairo, Egypt, and the M.S. and Ph.D. degrees in 1986 and 1991 from Caltech, Pasadena, CA, all in electrical engineering.

From 1985 to 1990 he was a Teaching and Research Assistant at Caltech. From September 1990 to July 1991 he held a Research Associate position at Texas A&M University. Since July 1991 he has been a Senior Research Scientist at QANTXX, Houston, TX. His research interests are the fields of neural networks, signal processing, forecasting theory, optimization theory, and pattern recognition. He has written over 30 publications in these fields.

Yaser S. Abu-Mostafa received the B.Sc. degree in electrical engineering from Cairo University in 1979, the M.S.E.E. degree from the Georgia Institute of Technology in 1981, and the Ph.D. degree in electrical engineering and computer science from Caltech in 1983, where he received the Clauser Prize for the most original doctoral thesis.

He joined Caltech as an Assistant Professor in 1983. He is currently an Associate Professor of electrical engineering and computer science, and a member of the Computation and Neural Systems faculty at the California Institute of Technology.

Dr. Abu-Mostafa received the ASCIT teaching excellence award three times, in 1986, 1989, and 1991. He is a current recipient of the Feynman-Hughes Fellowship (1990-). He was the founding Program Chairman of the annual IEEE Conference on Neural Information Processing Systems in Denver, and a founding member of the IEEE Neural Networks Council. He served as Associate Editor of the IEEE TRANSACTIONS ON INFORMATION THEORY, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, and the *Neural Networks* journal, and is currently a member of the editorial board of the *Journal of Complexity.*